Developer's Perspective on Containerized Development Environments — A Case Study and Review of Gray Literature

Ilkka Kuisma

Helsinki December 8, 2019

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Ilkka Kuisma

Työn nimi — Arbetets titel — Title
Developer's Perspective on Containerized Development Environments — A Case Study and Review of Gray Literature

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	December 8, 2019	49 pages + 0 appendices

Tiivistelmä — Referat — Abstract

Context: The advent of Docker containers in 2013 provided developers with a way of bundling code and its dependencies into containers that run identically on any Docker Engine, effectively mitigating platform and dependency related issues. In recent years an interesting trend has emerged of developers attempting to leverage the benefits provided by the Docker container platform in their development environments.

Objective: In this thesis we chart the motivations behind the move towards Containerized Development Environments (CDE) and seek to categorize claims made about benefits and challenges experienced by developers after their adoption. The goal of this thesis is to establish the current state of the trend and lay the groundwork for future research.

Methods: The study is structured into three parts. In the first part we conduct a systematic review of gray literature, using 27 sources acquired from three different websites. The sources were extracted for relevant quotes that were used for creating a set of higher level concepts for expressed motivations, benefits, and challenges. The second part of the study is a qualitative single-case study where we conduct semi-structured theme interviews with all members of a small-sized software development team that had recently taken a containerized development environment into use. The case team was purposefully selected for its practical relevance as well as convenient access to its members for data collection. In the last part of the study we compare the transcribed interview data against the set of concepts formed in the literature review.

Results: Cross-environment consistency and a simplified initial setup driven by a desire to increase developer happiness and productivity were commonly expressed motivations that were also experienced in practice. Decreased performance, required knowledge of Docker, and difficulties in the technical implementation of CDE's were mentioned as primary challenges. Many developers experienced additional benefits of using the Docker platform for infrastructure provisioning and shared configuration management. The case team additionally used the CDE as a platform for implementing end to end testing, and viewed the correct type of team and management as necessary preconditions for its successful adoption.

Conclusions: CDE's offer many valuable benefits that come at a cost and teams have to weigh the trade-off between consistency and performance, and whether the investment of development resources to its implementation is warranted. The use of the Docker container platform as an infrastructure package manager could be considered a game-changer, enabling development teams to provision new services like databases, load-balancers and message brokers with just a few lines of code. The case study reports one account of an improved onboarding experience and points towards an area for future research. CDE's would appear to be a good fit for microservice oriented teams that seek to foster a DevOps culture, as indicated by the experience of the case team. The implementation of CDE's is a non-trivial challenge that requires expertise from the teams and developers using them. Additionally, the case team's novel use of containers for testing appears to be an interesting research topic in its own right.

ACM Computing Classification System (CCS):
Software and its engineering → Software creation and management → Software development techniques

Avainsanat — Nyckelord — Keywords

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

The ”works on my machine” syndrome is a condition that occurs whenever something that worked on a developer's machine fails to run once it is executed in another environment (Humble and Farley 2010; Erfani Joorabchi et al. 2014). Causes for the syndrome are numerous but one common culprit is the lack of parity across execution environments (Spinellis 2012). This means that a missing library or environment variable, or any other difference across environments is enough to cause this issue.

The emergence of various practices and tools like continuous integration and automated deployment scripts have helped combat the effects of this syndrome (Humble and Farley 2010; Meyer 2014). The advent of Docker containers in 2013 provided developers with a way of creating immutable images of software bundled with all dependencies required to run the application (*What is a Container?* 2019). The promise of Docker is that when these immutable images are executed as containers, they will run exactly the same on any Docker Engine regardless of the underlying platform (Merkel 2014).

In recent years, there has been an emerging trend of leveraging the benefits provided by containers to combat the ”works on my machine” syndrome in development environments. We use the term Containerized Development Environment (CDE) to refer to the various implementations of development environments utilizing Docker containers in some capacity.

In this thesis we conducted a joint review of gray literature and qualitative single-case study with the aim of charting the motivations behind this trend as well as the benefits and challenges that developers claim to experience after taking containerized development environments into use. The study is comprised of three parts; in the first part we conducted a semi-systematic literature review to create an initial set of concepts of claimed motivations, benefits, and challenges. In the second part of the study we conducted semi-structured theme interviews with a small-sized development team that had recently taken a containerized development environment into use. In the third and final part of the study we compared the results from the two cases, in part to validate the findings and in part to enrich our interpretation of the results.

The underlying goal of this thesis is to lay the groundwork for further research by establishing an understanding of the current state of containerized development environments. We hope to uncover novel and interesting approaches that can be investigated in more detail in future work.

2 Background

2.1 Continuous Integration

Continuous integration introduced the idea of continually building and testing the codebase with every change to ensure that the software was never in a broken state (Humble and Farley 2010). In practice, continuous integration requires the use of version control which contains the entire codebase along with everything needed to build and run the software (Duvall et al. 2007; Meyer 2014). The current state of the software as it exists in version control is referred to as the mainline or trunk of the codebase (Duvall et al. 2007).

In order to verify that the software is in a functional state, continuous integration requires an automated build process along with a suite of automated tests. It is common practice to use a separate continuous integration server that builds and tests the software automatically on every change committed to the codebase in a separate continuous integration environment.

It is also commonly suggested that the continuous integration environment should be as close a copy of the production environment as possible. (Duvall et al. 2007)

2.2 Typical Continuous Integration Workflow

Continuous integration is characterized as a practice rather than a set of tools (Humble and Farley 2010; Rogers 2004). The typical development workflow when using continuous integration starts with developers checking out a copy of the mainline from version control onto their local machines. The software is then developed in the local development environment and is referred to as a working copy of the software.

Once the developer is done making changes to the code, along with any associated automated tests, the developer will update their local working copy with new changes that have been added to the mainline in version control. Once the changes have been consolidated, the developer builds and runs the software in their local development environment and upon passing commits the working copy to version control.

Typically this action of committing code to version control will automatically trigger the building and running of tests in a separate continuous integration environment. If the software is built successfully and the automated test suite passes, the committed code becomes the new mainline of the software. If the build fails or the tests do not pass, the developer must locate and fix the issues before continuing with any work.

2.3 Works on My Machine Syndrome

So far we have mentioned three different environments that the software runs on: the local development environments of developers, the continuous integration envi-

ronment, and the production environment where released versions of the software are deployed. Even with the use of continuous integration it is possible to witness defects in the production environment that do not appear in the local development environment. This is referred to as the works on my machine syndrome (Humble and Farley 2010). The syndrome not only exists between production and development environments but can also exist across the development environments of developers working on the same codebase.

2.4 Continuous Delivery and DevOps

Continuous integration deals with the development of software but not the deployment of software into production. Continuous delivery seeks to pick up where continuous integration left off, by addressing the bottlenecks experienced in the delivery process (Humble and Farley 2010). The goal of continuous delivery is to make sure that the software is always in a releasable state.

Continuous delivery is extended by continuous deployment, where each validated change that is made to the codebase is automatically deployed to production if it passes the required checks. In practice this is done by introducing a deployment pipeline — an automated collection of tools and processes for moving the software all the way from version control to the production environment through the appropriate stages of quality control and other operations.

Continuous deployment, continuous delivery, and continuous integration are closely related to DevOps, which can be characterized as a methodology aimed at bridging the gap between the development of software and its operations (Jabbari et al. 2016). Some definitions also emphasize that DevOps has a lot to do with mindset (Rajkumar et al. 2016) and that it prescribes a specific kind of DevOps culture (Davis and Daniels 2016).

2.5 Dependency Hell and Containers

One of the issues that continuous delivery deals with is dependency management. By dependencies, we refer to all the other pieces of software that are required in order to run the application — it is worth noting that dependencies in our use of the word refer to specific entities, and not a relation between two entities. Dependency hell is a problem that occurs when the developed application has specific requirements for dependencies that are missing or unmet when it is deployed to production (Humble and Farley 2010).

Container technologies such as Docker present a potential solution to this problem by packaging the components of the software along with their dependencies into isolated containers (Merkel 2014). By doing so, the packaged components of the software become portable and can be run on any system that runs Docker (*What is a Container?* 2019). Containers are a powerful tool that can be used in the

packaging and deployment stages of continuous deployment pipelines (Jaramillo et al. 2016).

2.6 Containers and the Docker Platform

Containers are characterized as lightweight counterparts to virtual machines, offering virtualization at the operating system level as opposed to the hardware level (Merkel 2014). Docker is a container platform that leverages existing technologies like LXC's (LinuX Containers) in order to create a tool that is geared towards the needs of developers (Fink 2014). Due to its wide adoption among developers, our discussion regarding containers will focus exclusively on the Docker platform.

The Docker platform allows developers to package their code along with the runtime, system libraries, and other configurations required to run the application successfully (*What is a Container?* 2019). Docker images become containers when they are executed on the Docker Engine, promising to run the same on any platform.

The Docker platform also consists of the Docker Hub which is a public registry hosting large amounts of ready-made Docker container images. Developers can find officially supported images for popular libraries and services like Nginx and MongoDB as well as runtimes for popular programming languages like Node.js and Python, and more.

2.7 Containers in Development Environments

Containers are commonly used in production environments, but there is a growing trend of using containers also in development: developers are running the components of the System Under Development (SUD) inside of containers on their local development machines. We use the term Containerized Development Environment (CDE) to refer to this new way of working, that is taking shape and starting to see wider adoption.

One key feature of CDE's is to move all of the dependencies of the SUD inside of containers. External components like databases are simple to take into use because ready-made container images for running them are available via Docker Hub, for example. Additionally, the proprietary source code being developed is executed in runtimes that are installed and configured inside of containers. Developers do not need to install any databases, runtimes, or other dependencies directly onto their local machines in order to be able to run the SUD.

The source code of the application being developed is stored and edited in the local file system on the development machine. The source code files are then shared with a container that executes the source code in the container's runtime. This sharing of files between the container and its host is accomplished by mounting the source code into the container as a volume or bind mount (*Manage data in Docker* 2019).

Container orchestration platforms assist the configuration and management of con-

tainer deployment in production (Khan 2017). The Docker Compose orchestration tool makes it possible to specify the configuration of multiple containers in one file, and start all of the containers with a single command (*Overview of Docker Compose* 2019). Docker Compose is featured in many CDE's for managing the configuration of the SUD on the development machine.

While there are many other implementation details related to CDE's, these are the most common concepts you see in most examples presented in blog posts and tutorials online. The purpose of this thesis is not to provide a technical overview of CDE's, but rather to chart the motivations behind the trend and the experiences people have had with using containers in their development environments.

3 Research Approach

3.1 Research Questions

This thesis seeks to form an understanding of the emerging phenomenon of containerized development environments by answering the following research questions.

RQ1. What reasons do developers state as the motivation behind moving to a containerized development environment?

RQ2. What benefits do developers claim to experience with containerized development environments?

RQ3. What challenges do developers claim to experience with containerized development environments?

By finding answers to these questions, we aim to create an overview of containerized development environments and to lay the groundwork for future research. The questions are intentionally formulated in a way to capture self-reported claims made by developers regarding containerized development environments. An additional goal of the study is to find reports of novel and interesting features, that can be followed up on and investigated in more detail in future work.

One natural follow-up to the research questions presented in this thesis, would be to ask what projects or situations the containerized development environment is best suited for — this thesis will contain some discussion related to this but it will not be the primary focus. Once an understanding of the motivations for containerized development environments has been established, one could also ask if other solutions exist for fulfilling similar needs and requirements.

In the following sections we present the research approach and methodology used in this thesis. The study is conducted in multiple parts, and we will begin with a description of the overall structure of the study. In the sections that follow, we describe each of the methods used in the different parts of the study in more detail.

3.2 Structure of the Study

The study is structured into three separate parts: a review of gray literature, a case study, and a joint discussion and comparison of the results obtained from the previous two. The literature review is conducted first to create an initial set of concepts that answer to the three research questions regarding motivations, benefits, and challenges. A qualitative single-case study is conducted afterwards to get a second data set that is compared against the results of the literature review. In the Discussion we take a look at the differences and similarities between the two sets of results, partly to validate our findings and partly to create a more detailed and richer interpretation of the results.

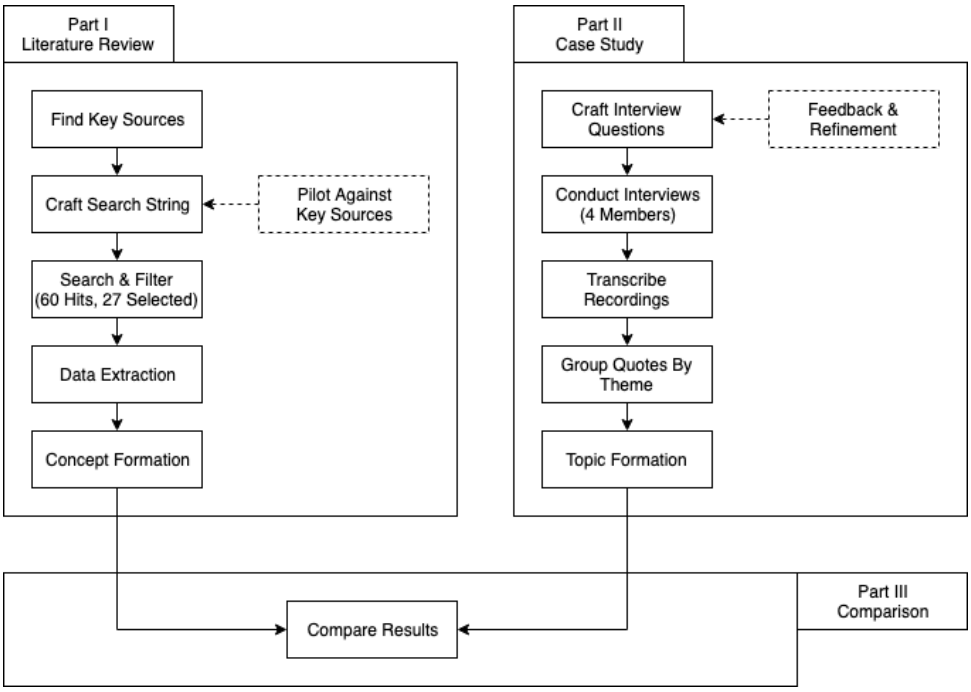

Figure 1: Overall structure of the study.

Various online blogs and forums contain discussion and reports about the opinions, claims, and experiences of individual developers working with containerized development environments, making it a suitable choice of literature regarding the scope of our research questions. We systematically conduct a review of gray literature by using Advanced Google Search to find relevant online sources, after which each source is stripped of quotes that speak of a challenge, motivation, or benefit. For each set of quotes we conduct free-form concept formation to identify the higher level topics related to motivations, benefits, and challenges.

In the second part of the study we conduct a qualitative single-case study by interviewing all members of a development team that had recently taken a containerized development environment into use. The data for the case study is gathered through semi-structured theme interviews that are recorded and then transcribed. The transcriptions are analyzed to identify higher level themes and topics that emerge during the interviews.

The purpose of the literature review is to capture many different opinions and experiences from a large number of different developers and projects, while the purpose of the case study is to gather highly detailed information from a team of developers working on the same project. In the third and final part of our study, we compare the results from the literature review and the case study, and report our findings.

3.3 Part I: Review of Gray Literature

3.3.1 Rationale for Gray Literature

We purposefully chose to source the material for our literature review from online forums and blogs commonly used for discussing the topic. These types of websites contain many reports about the opinions, claims, and experiences of individual developers working with CDE's, making it a suitable choice regarding the scope of our research questions. In addition to the relevance of the material, its high availability was another important factor — we wanted to have a large enough collection of opinions and experiences in order to create as complete an overview as possible.

3.3.2 Constructing the Search String

An initial free-form Google search was conducted to locate key sources discussing the issue. Based on these sources we were able to identify a handful of websites that were commonly used for discussing the topic:

1. https://www.reddit.com
2. https://www.medium.com
3. https://hackernoon.com

We used the Google Search functionality for restricting the search to the listed websites and constructed a search string that returned the key sources and others similar to them:

```
site:medium.com
  local
 AND
  development
 AND
  environment
 AND
  (container OR docker)
 AND
  (experience OR opinion OR thoughts)
```

The same search string was ran against all websites except Hacker Noon which required the removal of the final clause concerning thoughts and experiences in order to return the desired sources.

3.3.3 Inclusion and Exclusion Criteria

We came up with the following inclusion and exclusion criteria to select the sources to be included in this thesis:

1. The discussion must be no older than three years old.
2. The discussion has to contain some information specifically about claimed benefits, challenges, or motivations of developers using containers for local development.

The Docker platform has evolved over time and we wanted to form an understanding of the current state of containerized development environments. We defined the first criterion for this reason and the specific time-frame of three years was selected based on our set of key sources and initial free-form survey. The reasoning behind the second criterion was to filter out tutorials and other sources that do not contain relevant information.

Only the first 20 search results from each website were checked against the criteria in order to cast as wide a net as possible, while maintaining a manageable workload for data extraction.

After performing the search and filtering the results against our listed criteria, we ended up with a total number of 27 sources that are shown in Table 1.

3.3.4 Data Extraction

Each included source was read in detail and its contents were extracted with the form shown in Table 2 to find the relevant information pertaining to the research questions. The sources were stripped for quotes that either spoke of a motivation to adopt the use of a containerized development environment or were an explicit statement of claimed benefits or challenges after adoption. Even if positive statements appeared to be grounded in experience, they would be categorized as a motivation rather than a benefit if no explicit claim was made about having experienced the benefit in practice.

The extracted data were used to come up with a separate set of concepts for each three of the categories. First, the extracted quotes were collected and divided into three separate sets based on their category. The same process for concept formation was then repeated for each set of quotes.

3.3.5 Concept Formation

The review uses the concept-centric approach described by Webster and Watson (2002) and follows their recommendation for compiling concept matrices as we analyzed the extracted quotes. The final concept matrices presented in the Results

Table 1: Sources selected for the review of gray literature.

SID	Reference
S1	Using Docker Containers As Development Machines (2018, October 10) https://medium.com/rate-engineering/using-docker-containers-as-development-machines-4de8199fc662
S2	r/docker - Can you do ALL of your local development? (2017, September 20) https://reddit.com/r/docker/comments/718t1v/can_you_do_all_of_your_local_development/
S3	How to create consistent development environments that just work (2017, February 6) https://hackernoon.com/how-to-create-consistent-development-environments-that-just-work-55be5417341b
S4	Don't install Postgres. Docker pull Postgres (2018, September 4) https://hackernoon.com/dont-install-postgres-docker-pull-postgres-bee20e200198
S5	Introducing a Rails 5.1 development environment for Docker (2018, February 3) https://medium.com/@michiels/introducing-a-rails-5-1-development-environment-for-docker-a783f00ac908
S6	The Benefits of using Docker for Development and Operations (2017, September 28) https://medium.com/uptime-99/the-benefits-of-using-docker-for-development-and-operations-2c5256ad89bc
S7	The Advantages of Using Docker for Web Development (2018, February 19) https://codeburst.io/the-advantages-of-using-docker-for-web-development-23096c457fad
S8	Efficient development with Docker and docker-compose (2018, November 9) https://hackernoon.com/efficient-development-with-docker-and-docker-compose-e354b4d24831
S9	r/docker - Using Docker as a Development Environment? (2016, July 22) https://reddit.com/r/docker/comments/4u1zyf/using_docker_as_a_development_environment_anyone/
S10	Development Environments with Docker (2015, December 31) https://medium.com/on-docker/development-environments-with-docker-89657c7b4ea2
S11	Our development environment with Docker (2016, November 22) https://blog.dockbit.com/our-development-environment-with-docker-ca6868f436dd
S12	Docker Development WorkFlow — a guide with Flask and Postgres (2018, January 8) https://medium.freecodecamp.org/docker-development-workflow-a-guide-with-flask-and-postgres-db1a1843044a
S13	Why using docker in your local dev environment is probably not a great idea (2018, April 9) https://medium.com/canal-tech/why-using-docker-in-your-local-dev-environment-is-probably-not-a-great-idea-3836c6823d60
S14	r/docker - Docker as a stable development environment? (2016, February 6) https://reddit.com/r/docker/comments/44h608/docker_as_a_stable_development_environment/
S15	Dockerized Odoo Development (2019, January 27) https://medium.com/@reedrehg/easier-odoo-development-278bbaab38c8
S16	Develop in Docker: a Node backend and a React front-end talking to each other (2018, May 31) https://medium.com/@xiaolishen/develop-in-docker-a-node-backend-and-a-react-front-end-talking-to-each-other-5c522156f634
S17	Why and How to Use Docker for Development (2015, April 28) https://medium.com/travis-on-docker/why-and-how-to-use-docker-for-development-a156c1de3b24
S18	Why use Docker? 3 reasons from a development perspective (2018, October 11) https://hackernoon.com/why-use-docker-3-reasons-from-a-development-perspective-8f46cf68c864
S19	Docker workflow for React/Web applications (2017, October 25) https://hackernoon.com/docker-workflow-for-react-web-applications-b62b09571736
S20	An Introduction to Docker Through Story (2017, December 28) https://hackernoon.com/an-introduction-to-docker-through-story-8ae5594d7446
S21	How Docker Changed My Workflow (2017, December 7) https://hackernoon.com/how-docker-changed-my-workflow-b953b79b73ff
S22	r/PHP - Dockerize local development or not? (2017, April 28) https://reddit.com/r/PHP/comments/681ny3/dockerize_local_development_or_not/
S23	r/docker - Docker for development, why, and how? (2017, August 17) https://reddit.com/r/docker/comments/982cag/docker_for_development_why_and_how/
S24	How we happily dockerized our development environment (part 1/2) (2016, March 10) https://hackernoon.com/how-we-happily-dockerized-our-development-environment-part-1-2-b05fd6927a53
S25	r/devops - Local development environment, docker or Vagrant + VM? (2017, October 28) https://reddit.com/r/devops/comments/799co3/local_development_environment_docker_or_vagrant_vm/
S26	r/docker - Do you Develop in Docker or Dockerize your application after the fact? (2017, November 27) https://reddit.com/r/docker/comments/7fw0m8/do_you_develop_in_docker_or_dockerize_your/
S27	r/PHP - How to increase Docker performace for local dev on windows AND mac? (2017, March 31) https://reddit.com/r/PHP/comments/62kne2/how_to_increase_docker_performace_for_local_dev/

Table 2: Data Extraction Form, Literature Review.

Identification	
URL	Hyperlink to source
Primary Data (Collected as a list of items)	
Quote	Direct citation from source
Category	Motivation, challenge or benefit

follow the format of the examples presented by Webster and Watson (2002) with the addition of a column for the total source count per concept.

The quotes were examined in detail individually and were assigned to an initial concept. Each quote was first checked against all pre-existing concepts and assigned to one if it aligned with a pre-existing concept. If the quote did not match any of the existing concepts, a new concept was created and the quote was assigned to it.

Once the initial assignments were completed, the quotes and concepts were examined once more. If concepts were overlapping they were merged together, and those with different core ideas were separated. Finally, the extracted data were compared against the sets of categories to tally up the frequency of each concept.

3.4 Part II: Case Study

3.4.1 Case Selection

The case study portion of this thesis was conducted as a qualitative single-case study, where an information-rich case was purposefully selected for its convenience and practical relevance. The team and project that were selected for the study were relevant to the research questions and overall topic, as the project was one where the team had recently taken a containerized development environment into use. The composition and history of the team was known by the author of this thesis beforehand as he was a member of the case team working on the project. In addition to convenient access to the team and its members, this also provided us with additional insight pertaining to the relevance of the case.

3.4.2 Case Description

The software development team that was selected for the case, is part of the TOSKA research group at the University of Helsinki. The project that the team works on is an academic decision support system, aimed at providing real-time data visualization and analysis for decision makers and administrators at the university. The team had taken a containerized development environment into use in the project approximately half a year before the time of writing, and was actively using it.

The team was comprised of four developers and one manager. One developer had been part of the team for two years, two of the developers had been part of the team for one year, and one developer had been in the team for four months. The latest member of the team had joined the project after the CDE was already in use, and all others were part of the team during its adoption. The manager was the original founder of the research group and had recruited all developers of the team.

The team is self-organizing and all developers share the title of full stack developer in their job description. The team uses a modified lightweight form of Kanban for organizing its tasks, and does not follow any strict software development framework. The team is DevOps-oriented and embraces continuous integration, continuous delivery and deployment, containerization, and many other DevOps staples. At the time of writing, the system under design was transitioning from a traditional multi-tier architecture towards a microservices architecture.

3.4.3 Data Collection and Units of Analysis

The data collection method used in this study was semi-structured theme interviews, that were conducted with selected members of the team separately. We interviewed three of the team's developers and the team's manager, who are listed in Table 3. No additional material related to the team or project was collected.

Table 3: Identifiers and descriptions for the interviewed members of the team.

ID	Role	Joined Team	Description
I1	Manager	2 years ago	Founder of the research group.
I2	Developer	2 years ago	Did not participate in CDE implementation.
I3	Developer	1 year ago	Active member during CDE implementation.
I4	Developer	4 months ago	Joined after CDE was already in use.

We defined three different units of analysis for the case study: the individual developers, the development team, and the project itself. These units of analysis were selected to garner a holistic view of the impact that the containerized development environment had at different levels relating to the project.

The interview questions were drafted under the guidance of a senior researcher and were later reviewed by the same person for improvements and errors. The purpose of the interview questions was to initiate the discussion on the themes and the questions were designed to cover topics related to the different units of analysis. The final revised list of interview questions is as follows:

1. What is your background in the project?
2. What is your job description in the project?

3. How did you first learn about Docker containers?
4. Why did you start using Docker containers?
5. How would you describe the CDE in your own terms?
   What parts does it contain?
   What does it do?
6. Why was the CDE taken into use by the team?
7. How has the CDE developed and evolved over time?
8. What are your overall thoughts and feelings on the CDE?
9. What effects has the CDE had on your work?
   What is different?
   What has changed?
10. Has the adoption of a CDE required something of you?
11. Has the adoption of a CDE required something from your team?
12. Have there been any surprising experiences or effects with using the CDE?
13. Would you use a CDE again in another project or recommend it to others?
14. Is there anything you would like to add on the topic?
15. Is there a topic that was not addressed in the questions?

All of the interviews were held within the same week and lasted for half an hour each. The interviews were conducted in Finnish as all members of the team were native Finnish speakers. All of the interviews were held in person except for one that was done over Skype. The audio from all of the interviews was recorded for transcription purposes.

3.4.4 Data Analysis

Once the interviews had been conducted, the recordings were transcribed and data was extracted from the transcripts with the form shown in Table 4. The transcripts were then analyzed in two separate parts: in the first part we went through each of the extracted responses and assigned them to a higher level topic or theme that was discussed in the response. In the second part we took the responses for each separate theme and assigned a more detailed concept that was specific to that particular topic.

Table 4: Data Extraction Form, Interviews.

Identification	
Interviewee	Name of the interviewee.
Primary Data (Collected as a list of items)	
Question	Question that led to the response.
Quote	Citation from transcript.

4 Results

The results from the literature review and case study are presented in two separate parts. The results from the literature review are presented first and the results from the interviews second. This section concludes with a short summary and comparison of the key findings from the results.

4.1 Part I: Review of Gray Literature

After filtering the search results against our listed criteria we ended up with a total number of 27 sources that are shown in Table 1. We then performed the concept formation and analysis for each of the three categories.

4.1.1 What are the Stated Reasons Behind Moving to a Containerized Development Environment?

Twenty-one of the included sources contained some expressed motivation for adopting a containerized development environment. The concept matrix for these sources is shown in Table 5.

Fourteen sources expressed *cross-environment consistency* as a motivation, meaning the desire to have a development environment that would run consistently across different platforms and machines without platform-related issues:

> "This allows developers to work on a common container configuration that runs on the same OS and toolset, thereby eliminating cross-platform compatibility issues almost completely." (S1)

This ties in closely with *developer happiness and productivity*, a concept referring to the desire to improve the working experience of developers through automating common manual tasks and other means. The following source describes how time spent on environment configuration and general frustration could potentially be lessened by using containerization, thus improving productivity and developer satisfaction:

> "With the amount of time I spend screwing around with dev environments, it seems like an investment in Docker will be worth it, and I'm tired of the 'works on my machine' problem anyway." (S2)

The concept of *developer happiness* was among the most mentioned motivations (11 mentions) and statements regarding it typically contained powerful language indicating pains caused by consistency issues:

> "... the one thing that has brought me to the brink of insanity more than any other is dealing with development environments." (S3)

Table 5: Expressed motivation behind moving to a containerized development environment. The references for the source identifiers are defined in Table 1.

Concept	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	Total
Cross-Environment Consistency	x	x	x			x	x			x		x	x	x	x	x	x	x	x			14
Developer Happiness and Productivity		x	x	x			x		x	x	x				x	x		x		x		11
Simple Setup	x			x	x		x			x	x				x		x			x	x	10
Built-In Dependency Management	x		x	x		x										x	x			x		7
Onboarding	x						x			x	x	x		x				x				7
Disposable Environment		x	x		x				x												x	5
Production-like Environment						x		x					x							x		4
Local CI	x											x										2
Enable Experimentation							x										x	x				2
Built-In Configuration/Tooling								x														1

Many of the statements that expressed the desire to improve *developer happiness and productivity* attributed it to a *simplified initial setup* process. The following source describes how the entire setup of the development environment can be done with just a few commands:

> "Installing software is hard. And it has nothing to do with your expertise as a developer ... Docker provides a way out of this mess by reducing the task of installing and running software to as little as two commands..." (S4)

The *simple setup* concept is tightly linked to and enabled by *built-in dependency management*, meaning the ability to automate the task of installing and configuring dependencies and make it a part of the initial setup:

> "In theory, developers would only need to download Docker and a text editor of their choice, and not have to install external tools and dependencies." (S1)

The *simple setup* concept was considered to help with and speed up the *onboarding* of new developers to teams and projects:

> "We also wanted to simplify the initial setup process for all our applications across the board. This will speed up the onboarding process for new engineers who join our team." (S1)

Five sources were motivated by having easily *disposable environments* and not having to install shared dependencies on developers' workstations. The following source describes this as being desirable as it makes it easier to reset the environment for the different applications being developed on his workstation:

> "Having a development environment per app also helps cleaning out any databases and Sidekiq queues and gem interdependencies. So you always have fresh and clean environment and you don't have to recompile Ruby for each differing version on your workstation." (S5)

Developers and teams that used containers and Docker for the deployment of their projects also mentioned the explicit motivation of developing the *exact same artifact that would be deployed to production*:

> "When you have your application in a Docker container, you can be sure that the code you're testing locally is exactly the same build artifact that goes into production." (S6)

Among the lesser mentioned motivations for adopting the CDE was the ability to run continuous integration tests locally:

> "This gives developers the ability to conduct on-demand integration tests by spinning up containers for the required services, satisfying our 3rd requirement. Allow developers to conduct independent integration tests." (S1)

Two sources mentioned that the containerized development environment would *enable experimentation* of different services and their versions:

> "Another example to consider is if you want to check how different versions of a library affect page rendering speed for your web interface. You can simply install different versions of the library in different instances of your front end container while keeping the application and database containers the same." (S7)

Lastly, one source mentioned the ability to add *built-in configuration and tooling* to the development environment as a reason for adopting the CDE. The source provides the example of adding tooling and configuration to create automatic test execution into the environment:

> "The result of this (setup) is that every code change will now automatically execute all tests... This makes test-driven development with Docker trivial." (S8)

4.1.2 What Benefits do Developers Claim to Experience with Containerized Development Environments?

Thirteen of the included sources expressed some claim about benefits with containerized development environments. The concept matrix for these sources is shown in Table 6.

Four sources mentioned the benefit of *cross-environment consistency* after adopting a containerized development environment. The following source claims that the unified development environment has helped decrease consistency issues:

> "Our shop very successfully uses Docker for providing a unified development environment that makes it much harder to say 'Works on my machine'." (S9)

Many sources (9 mentions) made a claim about having experienced the benefit of achieving the *simple setup* in practice. The following source demonstrated through a tutorial how setting up the example environment was possible with two commands:

Table 6: Benefits that were claimed to have been experienced after moving to a containerized development environment. The references for the source identifiers are defined in Table 1. Concepts are listed in order of appearance in the text.

Concept	S1	S4	S7	S8	S9	S10	S11	S22	S23	S12	S13	S24	S25	Total
Simple Setup	x				x	x	x	x		x	x	x		8
Infrastructure Package Manager			x		x			x	x	x		x	x	7
Shared Configuration & Tooling				x	x	x		x	x	x		x		7
Cross-Environment Consistency		x			x			x	x			x		5
Developer Happiness/Productivity		x				x		x				x	x	5
Disposable Environment		x						x	x					3
Hosting External Applications Easily	x							x		x				3
Onboarding							x							1

> "The result of all of this is a local development environment that on your computer with a git clone and running and continuously iterating from the moment you type: docker-compose up -d." (S10)

Only one source mentioned improved developer *onboarding* in practice. The source claims that new developers have been able to quickly setup a working development environment due to the *simple setup*:

> "The setup described above has enabled any new developer joining our team to have things running locally with a single command in under 10 mins." (S11)

Four sources contained a claim of improved *developer happiness and productivity* after adopting the use of containerized development environments. The following source claims that using Docker was the most productive thing they had recently done and highlighted the importance of environment setup and distribution:

> "Using Docker was one of the most beneficial and productive things I did in the past couple years. ... Docker helps me a lot with environment setups also environment sharing..." (S22)

Disposable environments that require no installation of dependencies directly on the developer's machine had been mentioned as a benefit in three sources. The following excerpt describes one developer's claims of being able to easily setup and dispose of virtual environments:

> "You can spin up whole stacks of virtual servers that work together- like a whole MEAN stack, or a whole ELK stack, in under an hour... And trash it just as fast if you don't like it." (S23)

The previous claim also brings up the benefit of using the Docker platform as an *infrastructure package manager*. Eight sources in total mentioned this benefit of being able to provision and experiment with different services with ease, and then distribute them across the team. The following source illustrates this benefit by claiming that adding Redis to the project environment could be done easily with a few lines of code, and that his team could recreate the same environment by rebuilding the container images of the CDE:

> "I could add a couple more lines and have a full production setup with Nginx and Gunicorn. If I wanted to use Redis for session caching or as a queue, I could do that very quickly and everyone on my team would be able to have the same environment when they rebuilt their Docker Images." (S12)

Three sources highlighted the benefit of using the CDE specifically for the management and distribution of external services like databases and message brokers. The following source claims that based on their experience, the containers in CDE's are best suited for hosting these external services (applications) locally:

> "What we have realized after all the trial and error is that Docker containers are best suited for developers to run self-hosted applications quickly and easily. Developers can then test their code by connecting to these local instances instead of connecting to remotely hosted instances." (S1)

The benefit of *shared configuration and tooling* was mentioned in 7 sources. In the following excerpt one developer explains that the different configurations of services can be included as a part of the project's repository and then shared across the development team:

> "...you can commit all these interwoven hardware emulations to your source code repository as text files, to be shared and replicated identically to your dispersed team worldwide." (S23)

4.1.3 What Challenges do Developers Claim to Experience with Containerized Development Environments?

Twelve of the included sources expressed some claim about a challenge that was experienced in the adoption of containerized development environments. The concept matrix for these sources is shown in Table 7.

Two sources mentioned *developer frustration* that was experienced as a result of the compounding of minor issues. The following source exemplifies this by claiming that the compounding ends up deteriorating the developer experience:

> "...this caused a lot of little inconveniences which when taken individually aren't that big of a deal but if you pile them together you get a really deteriorated development experience." (S13)

Table 7: Challenges that were claimed to have been experienced after moving to a containerized development environment. The references for the source identifiers are defined in Table 1. Concepts are listed in order of appearance in the text.

Concept	S1	S2	S3	S9	S10	S11	S22	S12	S13	S26	S14	S27	Total
Implementation	x		x	x	x	x	x			x			7
Decreased Performance	x					x	x		x	x		x	6
Requires Docker & DevOps Skills		x	x		x			x		x			5
Non-Linux Performance Issues							x			x		x	3
Issues With Volumes									x	x		x	3
Developer Frustration							x		x				2
Debugging Difficulty							x		x				2
IDE Integration & CLI Tools				x			x						2
Not For Desktop & Embedded											x		1

Six sources mentioned challenges that were experienced due to *decreased performance* when running software inside of containers. The following source claims that common commands and tasks in their development process doubled in execution time:

> "...every command took almost twice as long to complete. From eslint verification to our unit test suite every task seemed to take forever compared to before docker. This caused the apparition of unhealthy behaviors in our development team..." (S13)

Three sources claimed the existence of *performance issues on non-Linux platforms* specifically. The following source goes as far as to claim that CDE's are unusable for developers who do not use Linux as their development platform:

> "...that's why volumes are 60 times slower in Mac and Windows when you do that. It's basically useless for devs on Windows and Mac OS X." (S26)

Issues with volumes both in terms of performance and crashing were mentioned in three sources. The following source provides an example of how the switching of branches caused many files to change in the volume which then lead the application to crash:

> "...switching from one branch to another would cause tens of files to change at once and could make our Docker crash into a state where we weren't able to start the application again." (S13)

Debugging difficulties were mentioned in two sources which shared some relation with claims about challenges in *IDE integration and the use of CLI tools*:

> "Debugging is now harder. If your IDE has to connect to the interpreter, then you need a way to tell it to use the interpreter within the container to execute the code." (S22)

Seven sources claimed challenges that were experienced in the *implementation* and configuration of the containerized development environment. The following source elaborates on this by saying that constructing a CDE requires both knowledge of the tool and an understanding of what is suitable for the needs of the project:

> "There is no Docker way to construct a development environment. Docker is a composable tool, not a holy book. Instead of trying to copy someone else's Docker based build system, take the time to learn the tool, meditate on your needs, and then create an environment where you've used Docker to reduce your pain points." (S10)

The challenges related to *implementation* were closely tied to the challenge of acquiring the *required Docker and DevOps Skills* for the development team which was mentioned in five sources. The following source claims specifically that familiarity is required not only in Docker, but Linux and DevOps in general:

> "The only potential downside with Docker, is that to feel comfortable using it, you need to not only learn how Docker works, but you need to have some level of familiarity with Linux and DevOps concepts like containers, networking, etc." (S3)

Lastly, one source mentioned the limitation of containers in the development of *desktop and embedded systems.* The source claims that since the application deployment in these contexts is not done with Docker, there is nothing to gain in containerizing the application:

> "Desktop and embedded work don't have deployment under Docker. If I'm doing Windows development I have to build an installer and launch a VM to make sure it works. If I'm doing embedded work I need to reflash my hardware using our manufacturing process. Nothing is gained on the deployment side." (S14)

This concludes the literature review portion of this study. The results brought forth in this and the following section covering the case study are summarized briefly in Section 4.3 and discussed further in Section 5.

4.2 Part II: Case Study

Upon analyzing the interview data we identified eight different topics. Some of the topics emerged as a result of the interview questions and others emerged naturally

during conversation. The topics ranged from the history of how and why the CDE came about (4.2.1), to the experiences the team members had with the CDE (4.2.4 – 4.2.5), to a more general evaluation of what the CDE required from the team (4.2.6) and what would make it a suitable choice for a team or project (4.2.8). We will go through the interview results topic by topic, in the aforementioned order. Excerpts from the interview transcripts will be included for each topic and the interviewees will be referenced by the identifiers defined in Table 3.

4.2.1 Background and Creation

The containerized development environment was not adopted by the team in a single event, but rather the move towards it happened step-by-step, as was noted by the project manager of the team. One of the longer standing employees states that when he joined, containers were used exclusively in production:

> "Originally only the production system was run using containers and we did not really use them locally." (I1)

Containers were first introduced into the development environment for running external services like databases. The longest standing member of the team pointed out that there had been an attempt to create a containerized development environment early on in the project before the other developers had joined:

> "We had two ways of running the system, either locally or in containers. I think that we had some technical issues with using containers, and we decided to move forward with the project instead of trying to fix the issues, which is why we abandoned it in the beginning ... I feel like at the time it was more about learning to use Docker than anything else." (I2)

The next step towards the CDE happened when the team introduced a new service along with its external dependencies into the system. The team created a new CLI tool to help with the setup of the project:

> "...something was difficult, and we started to work on the CLI that would automatically setup the development environment at the click of a button. This was because there were so many moving parts that the manual setup was laborious and painful." (I3)

The CLI did not initially run the entire development system in containers. Only the external dependencies were set up and run with Docker, and the other services were set up with custom scripts. The motivation to fully containerize the application, was to get rid of the overhead related to maintaining these scripts:

> "... that gave us the idea, that instead of trying to get the Bash scripts to work on everyone's machine, it would just be smarter to do it all with Docker." (I3)

New issues and pain points emerged after the move to a fully containerized development environment was made. The project manager recalled that one member of the team felt, that they would have to either fully adopt or abandon the CLI:

> "I remember that one of the members of our team kept saying that the 'CLI doesn't work'. He also brought up the idea, that we should make the decision to either abandon the CLI or switch to it exclusively." (I1)

The team made the switch to the exclusive use of the CDE version of the CLI, and the issues related to it were eventually fixed. Members of the team who had initially experienced problems, said that now the CLI works to the point where they almost do not remember the issues. The CDE has continued to evolve after its initial adoption, partly as a result of moving towards a microservice architecture:

> "...new parts of the system are now implemented as new services running in their own containers, and this is the general direction of our application." (I4)

The team had also added developer tools to the CDE that were configured and automatically setup in every development environment:

> "We have also added new tools like Adminer, that we use for accessing the development databases from a web UI ... before we had to access the database from the command line." (I4)

The team had also built their new end-to-end testing strategy around the containerized development environment:

> "This new way of doing end-to-end testing with Cypress relies heavily on it (the CDE)." (I1)

4.2.2 Motivations and Drivers

The containerized development environment came about when the team made its first step towards a microservices architecture as was explained by the members during the interview. The increase in services and the particular effect it had on the setup of the development environment were mentioned as a primary motivation and driver for the emergence of the CDE:

> "Especially now, when we are adding new services like Apache Kafka to our system, setting up the environment would get out of control if we had to install them manually." (I1)

This was also viewed from the perspective of the CDE being an enabler for the move towards the microservices architecture, and the increase in services appeared to be what was creating the need for a simpler setup:

> "... it is possible that only after taking the CDE into use, have we have been able to move towards the microservices architecture in our system, since it makes it so much easier to have all of the services configured and running together with one command." (I2)

Another motivation that was expressed by all members, was the desire to have a shared baseline and a consistent environment across the team:

> "I think that the point was that we would all have the same development environment and baseline, and that we would get rid of situations where something worked on someone's machine but not on someone else's ... so if something did not work, we could all work on the problem and then know exactly what needs to be done to get it working for everyone." (I3)

The team's project manager also brought up the point, that from his perspective it is important that the team's workflow fits the needs of the team, and that this type of process improvement is exactly the kind of thing that the team should be doing.

4.2.3 Key Features

When asked to describe the CDE in their own terms, all members mentioned similar key features. One common thing that was mentioned, was that it contains all of the different parts of the system needed to run it:

> "It contains all of the language runtimes and external dependencies and services. It contains whatever services it's attached to ... if you use an external logging service like Sentry, then you can easily pull an image for Sentry from Docker Hub and add it to the system." (I4)

In addition to consistency across developers' environments, the fact that the development environment mimics production was also brought up:

> "One really big thing is that our development environment closely matches our production environment ... that way we can be sure that we are using the same versions of our system's dependencies everywhere." (I2)

One thing that was brought up was the isolation that the CDE offers, both across services and also from the host running the service containers:

> "It isolates my development environment from my development machine's settings ... the fact that it was isolated meant that all my development Python environments worked, even when the Python setup that was running on my own machine was completely broken." (I3)

The last key feature was that the configuration of different parts of the system was shared and also explicitly defined:

> "It clears up our dependencies and configuration, you can go and see what the different parts need, so that it's not all on your computer but that every container has its own configurations and dependencies defined." (I4)

4.2.4 The Experienced Benefits

Interviewees brought up benefits they had experienced at various points throughout the interview. One benefit that emerged was automatic updates to the development environment, that did not require any additional work from the developers:

> "It hasn't really required anything from me personally ... I've just pulled the latest version and then everything has just worked." (I2)

Some of the developers emphasized that the CDE helped with managing the additional complexity of dealing with microservices. Especially provisioning new services, and starting up as well as tearing down the environment:

> "... in a microservices environment it makes so many everyday tasks like starting up the entire system much easier, and especially adding new databases is just so simple..." (I2)

The team also brought up the importance of configuration management, which was viewed from a few different angles. One benefit they saw was that the explicit definition of how the system is configured in a single file brings additional clarity to how things work in the system. Additionally, the composability and configurability of the environment was considered important:

> "It helps with setting up new services, their communication, and it makes it so much easier to understand where the different parts of the system belong. It's even easy to add multiple databases to the system. You just add a few lines of code to a file and you have a new database up and running." (I4)

One of the developers saw that the value came from the consistency it created across the environments of developers:

> "It just gets rid of any conflict and discrepancies between environments. Since we have the containerized environment we know that everything is actually configured the exact same way." (I3)

Consistency between development and production was also emphasized by another developer, who saw it as important based on his experiences from previous projects and workplaces:

> "Our customer had a different version of Java running on their server than what we had, and it lead to the entire application not working in production. That was actually because of a minor update to Java and not a major one. There is definitely a benefit in having the same environment everywhere." (I4)

Another benefit that was experienced by developers was the knowledge related to Docker that they had acquired in the process. The interviewees felt that the adoption of the CDE had not only required them to learn more about Docker, but that using it actively during development had also taught them more about it:

> "... it has expanded my understanding of how Docker and Docker Compose work on a very detailed level. The things that I have been able to get out of them after using them, has been a really positive thing." (I2)

Two interviewees stated that they were surprised that Docker could be used in such a way, and they were also surprised to see that the concept works in practice:

> "I'm really surprised that the thing works to begin with. I had no idea that you could use Docker this way..." (I3)

One developer expressed that being forced to learn more about Docker was positive because it made the team acquire skills that were useful for deployment and dealing with the production environment:

> "The fact that you have to use Docker more often and touch the more difficult configurations is a positive thing in the grand scheme of things, since we use Docker in production and it just benefits our team if everyone knows how to use those production tools better." (I2)

The same developer also expressed that deploying new services to production was slightly easier, because adding the service to the CDE was done in a similar way as it would be done in production later on:

> "Adding any new services is easier, because you have to think about how it is configured and how it actually runs in production, and they work the same way in terms of what environment variables need to be configured and so forth." (I2)

The case team had also leveraged the features of the containerized development environment in their implementation of end to end testing. The team ran their browser-based user acceptance tests against a slightly tweaked version of the containerized development environment, and the same mechanism of setting up and running the tests was used both locally and on the continuous integration server. One of the interviewees also saw potential in using containers as a means of running a large amount of tests efficiently in parallel:

> "I've seen tests in the past that had to use the same shared database. If instead you had this kind of Docker environment then you could spin up multiple instances of the same service and run different tests against them. You could basically do parallel testing and it wouldn't matter how slow the individual tests suites are because you could run multiple suites at the same time." (I4)

4.2.5 The Experienced Challenges

Most interviewees had experienced the same set of challenges in the adoption and use of the containerized development environment. One of these challenges was that the CDE introduced changes to the team's previous workflow. An interviewee mentioned that developers now have to remember to run commands inside of containers and that running tests currently relies on containers:

> "Sometimes I try to run npm install on my own machine and wonder why it doesn't work. It takes like half an hour to remember that the npm install command needs to be run inside of the container ... and another thing is that you need to have the containers of the application running in order to run tests." (I3)

The changes to the workflow are something that caused the project manager to opt-out of using the CDE at the beginning:

> "... my previous development environment just worked and my debugging workflow was optimized for it. It just worked and I didn't want to fix what was not broken." (I1)

All interviewees brought up the difficulty and challenge of getting development tools like IDE's and linters working out of the box. It appears that it is possible to get them working, but that it is not straightforward:

> "One drawback is that debugging tools for Node require some kind of extra configuration in the development environment to get them working. All of the instructions that exist for those tools assume that you are running the Node runtime on your local machine, but in our system it's inside of the container." (I4)

The difficulty in configuring tools like linters, that were a part of the team's regular workflow, presented a challenge in the implementation of the containerized development environment:

> "I don't remember what the issue was, but getting npm and the linters configured correctly was a huge obstacle and nothing was working for a quite some time." (I2)

The project manager of the team made the observation that it was quite common to assign new tasks for fixing the CLI, and that it took a while to get it to the point where it was working properly. The difficulties in the implementation of the containerized development environment was seen as presenting an additional challenge of eating up development resources from the team:

> "Getting to the point where it was working ate up a lot of our time and resources. Since we don't have any strict demands on how many features we have to deliver this was acceptable in our context. I do find myself worrying about the balance between creating value for the customer and improving our own workflow, and that was something I kept thinking about." (I1)

4.2.6 What its Adoption Required

There was some discussion in the interviews about the things that were required from the team in order for the CDE to become a reality, and more general discussion about what it would take from another team to adopt it. The manager of the team observed that it required a certain skill-set and a vision from the team:

> "It required that the team had the vision and necessary skills to make the decision that this would be an improvement over what we have now." (I1)

The ability of the team to self-organize and the management style of the team were seen as fundamental in enabling the team to pursue the change. Also the team's commitment to continuous improvement was hinted at by the project manager:

> "You have organized yourselves and it has required me to be wise enough not to deny it ... this is exactly the kind of thing the team should have been doing. The old best practice was to have a good README, and this replaces that." (I1)

The developers of the team also saw the importance of having the kind of management and resources that supported and permitted the move towards the CDE:

> "Every developer in our team has a lot of opportunity to affect how we work as a team and the tools that we use for development. The fact that we have resources for this kind of experimentation is what makes it possible." (I2)

One of the team members saw that it was important for the team to have the necessary tenacity and grit to work through the challenges that they faced in the beginning:

> "It required the mental fortitude and grit to tackle the issues one by one in order to get to the point where everything worked. It would have been easy to scrap the idea and return back to the old way of working ... Our team needed to have individuals who were willing to go the distance and be patient enough to fix the issues. I think that really required a vision of what it would provide for us if we did get it working." (I2)

All members of the team felt that taking the CDE into use requires all of the members of the team to have knowledge of Docker and invest resources into learning it. One of the developers described the situation like an investment:

> "Managing dependencies is easier with Docker but it requires you to know how to use it ... it does require you to know about it and actually want to use it." (I4)

While learning Docker was seen as a requirement, most interviewees mentioned that in everyday use there are only a few commands you have to remember. The need for deeper knowledge arises when you want to change the configuration or add new services:

> "I would claim that anyone could use it if you just said 'write this command before you start developing'. I don't think that it requires any real knowledge of Docker unless you are setting up new services." (I3)

4.2.7 Views and Experiences of Onboarding

The manager of the team viewed the CDE as having major implications for onboarding new developers into the project. During the interview he expressed his vision for how the onboarding process and first day at work would go for a developer:

> "My definition of done is that when a new developer joins the team he tells the CLI to install the environment and then goes to get a cup of coffee. When he comes back from the coffee break, the development environment is setup and he then tells the CLI to run tests locally. Before lunch, the new recruit pushes his first commit to version control and when he comes back from lunch, the new version has been tested on the CI server and it has been automatically deployed to production." (I1)

One of the interviewed developers had joined the project after the adoption of the CDE and was able to share his experience of the onboarding process, as well as difficulties he has experienced with project onboarding in the past:

> "In previous jobs I have had to be spend a day or two in configuring the environment ... we actually had a list of the versions of different libraries that were compatible with the application and you had to manually find and install the correct version." (I4)

When describing his experience in the current project, he mentioned that setting up the environment was much easier than at previous jobs, and that it was really helpful that he had a course immediately available for learning Docker:

> "Setting up the development environment was painless, especially since Docker takes care of all of the networking and configuration. We also have many services and I didn't even have to start those separately, one command got them all up and running ... It was also a huge help that there was this new Docker course at the university that I could just take immediately. That was enough for me to learn what I needed to get started." (I4)

4.2.8 Suitability for Projects and Teams

The interviewees were asked in different ways to evaluate the suitability of the CDE to different projects. Two interviewees mentioned that if configuring the system becomes a major pain point, then the CDE can help in that situation. Having three to four services was mentioned by many as a critical point:

> "If there's many services or lots of complicated configuration that's done in many different places, then this is a good way of structuring that configuration into one place ... I think the critical number is when you three or more services, at that point it's worth it." (I2)

One of the developers mentioned that some particular languages are more difficult to configure than others, and could potentially benefit from the CDE:

> "If you have many team members and use languages that can get tangled up with your system configuration easily, like Python, then I would recommend it." (I3)

Many of the interviewees also pointed out that it might not be suitable if the project is short-lived, or if it is more important to dedicate resources to other things:

> "In another project I was working on we only had three months to get the application ready, and we would definitely not have wanted to spend a few weeks on trying to get the containerized development environment set up and working." (I2)

It was also pointed out that the CDE is mostly geared towards web services, and that it does not really offer any benefits for developing native mobile or desktop GUI applications.

4.3 Summary and Comparison of Results

We will close with a short summary and comparison of the key concepts and findings found in the review of gray literature and the case study. This summary is presented in Table 8 and the findings will be discussed in more detail in the next section. Some of the smaller findings in the results are not included in the final discussion in order to maintain the focus on the most prevalent findings.

The concepts formed in the literature review and the topics from the case study were formed independently. The summary presented in Table 8 does not attempt to find an exhaustive mapping between the two, but rather seeks to find the most relevant topics that were discussed, and then present what the case study and the literature review revealed about the topic. The relevant topics did not need to be present in both portions of the study in order to be included in this final comparison and discussion.

Table 8: Summary and comparison of the key findings from the literature review and case study. The general concepts listed in the first column are different from the concepts in the literature review.

Topic of Discussion	Review of Gray Literature	Case Study
Simple Setup	Having a one-click setup was one of the most mentioned motivations that was also experienced as a benefit in practice.	The need for a simple setup was driven by the increase of services and external dependencies in the case project.
Consistency	Consistency across development machines and between production and development was a common motivation that was also experienced in practice.	The team expressed the desire and benefit of having a shared baseline development environment that resembled the production environment.
Pain Points	Pain points experienced by developers with consistency issues and setting up environments were a commonly expressed driver for achieving consistency and a simple setup.	The increase in complexity of the case project lead to pain points in its setup that created the desire for an automated environment setup process.
Onboarding	Improved developer onboarding was a common motivation for a simple setup, but no experiences of developers being onboarded with a project using CDE were found.	The project manager saw implications for developer onboarding and one developer had experienced an improved onboarding experience when joining the project.
Isolation	Isolation between environments was reported to have the benefit of being able to easily switch between different projects and to reset the environment if something broke.	Services using Python were experienced as having troublesome environments that were easier to manage when they were isolated in their own containers.
Infrastructure Package Management	Commonly expressed benefit of using the Docker platform as a package manager for specifying, installing, and configuring infrastructure and services like databases.	Adding new services and infrastructure to the project could be done with just a few lines of code and distributed team-wide. This made it easier to add services and facilitated the move towards microservices.
Shared Configuration & Tooling	Reported benefit of having all configuration specified in a single file that was shared team-wide in version control. Some sources reported additional built-in tooling in the CDE that facilitated a specific workflow.	Experienced benefit of shared configuration and a document that defines the structure of the system. Additional use of ready-made images for browser-based development tools, e.g. for database management.
Performance Issues	One major challenge with CDE's was the decreased performance of the SUD running on the development machines, particular on non-Linux platforms. This had a negative reported impact on developer productivity.	Not mentioned in the case study.
Construction	One major challenge was the difficulty of constructing the CDE. The successful construction was reported to require more than just basic knowledge of containers, and the environment itself.	The technical difficulty of constructing the CDE was a struggle for the case team, and there was an additional reported motivational challenge to not give up on the CDE during its construction.
Resources & Support	Not mentioned in the literature review.	Case team mentioned that CDE implementation used up a lot of development resources from the team. Support and consent from the management was seen as important.
Tools & Workflow	Reported challenge of not being able to use certain development tools due to the execution environment existing inside of a container and not the host machine.	Issues with certain development tools not working without additional configuration. Additional challenge of having to change existing familiar workflows of developers.
End to End Testing	Not mentioned in the literature review.	The case team experienced the benefit of using the CDE for setting up a production-like version of the SUD that end to end tests could be run against. Suggested potential of being able to have multiple instances of the SUD for running end to end tests in parallel.

5 Discussion

In this paper we sought to find benefits and challenges developers claim to experience with using containerized development environments and what the stated reasons behind taking them into use are. We will briefly describe our findings for each of the three research questions before presenting a more detailed discussion of the twelve key findings of this study.

In our first research question we asked what reasons developers state as being the motivation behind moving to a containerized development environment. The most common reasons we identified in our review of gray literature were (1) wanting to provide a consistent development environment for all developers, (2) wanting to achieve a simple "one-click" setup process for creating a new development environment that would be easy to dispose of and reset, (3) to increase developer happiness and productivity by automating environment setup and by mitigating the effects of the "works on my machine" -syndrome, (4) to obtain built-in dependency management and automate the task of installing and configuring dependencies, and (5) to help developer onboarding through the simplified "one-click" setup.

The main driver we identified in the case study for adopting a containerized development environment was the case team's need to deal with the increased complexity and difficulties that followed as the number of services in the system under development began to increase. The interviewees stated that obtaining an automated "one-click" setup process and creating a consistent shared baseline for the development environment were the two most sought-after features.

In our second research question we asked what benefits developers claimed to have experienced with containerized development environments in practice. Many of the concepts that were mentioned as reasons for adopting a containerized development environment such as the simple "one-click" setup, increased consistency, disposable environments, built-in dependency and configuration management, and improved developer happiness and productivity were all claimed to have been experienced in practice in multiple sources. Only one source expressed a claim about improved onboarding in practice and three sources emphasized positive experiences with using containers specifically for running the external services, e.g. databases, of the application in development environments.

The members of the case team claimed to have experienced benefits similar to those that were identified in the review of gray literature, such as increased consistency across development environments, the automated "one-click" setup, built-in dependency and configuration management, and having a production-like environment in development. The members of the case team also made claims about benefits that were not present in the literature review. The interviewees claimed that using the CDE had improved the team's skills and knowledge of Docker and that the "one-click" setup made it possible to easily create instances of the system under development that automated tests could be run against.

In our third research question we asked what challenges developers claimed to

have experienced with containerized development environments. The most common claims about challenges that were identified in our review of gray literature were the difficulty in constructing the containerized development environment, the decreased performance of running the SUD inside of containers, and the amount of knowledge and skills required to successfully implement and use the containerized development environment. Three sources claimed that CDE's running on non-Linux development machines suffered from particularly poor performance, and two sources claimed that the compounding of minor issues they had experienced with using CDE's resulted in a greatly deteriorated development experience.

The case team made similar claims of having experienced challenges in constructing the containerized development environment. Many of the interviewed members pointed out that it took a major commitment from the team to fix all of the issues that came up during implementation. Members of the case team also claimed to have experienced challenges with getting command-line based development tools that were a part of the team's regular workflow to work with the containerized development environment.

After comparing the results obtained from the review of gray literature to the results of the case study, we were able to identify 12 key findings. The following sections contain more detailed discussion on these key observations made from the results of this study, as well as a comparison of the findings and methodology of our study to existing literature. This section will conclude with discussion about threats to validity and the limitations of this study.

5.1 Simple Setup and Consistency

Cross-environment compatibility as well as the desire for a simple setup process for the development environment were among the most mentioned motivations for adopting a Containerized Development Environment (CDE) in the literature review. These motivations and desires appeared to be met successfully according to the claims that developers made about their experiences, indicating that CDE's may provide a real-life solution to these problems.

Similar sentiments were echoed in the case study. The desire for a simple setup came about as the number of services in the System Under Development (SUD) started to increase. Many interviewees also mentioned that the CDE had mitigated discrepancies between developers' machines in practice, and that it helped to establish a shared baseline for the development environment.

Both the literature review and case study also reported a similar benefit and motivation of having a development environment that closely resembled the production environment. Interviewees in the case study mentioned that this provided increased confidence in deploying new changes to production. It is worth noting that this does not constitute hard evidence of compatibility related issues being mitigated in the deployment process, but it is unlikely that this confidence would be evident if complications in deployments had worsened.

There appears to be a commonly shared desire to mitigate compatibility issues and automate the setup of development environments. This desire is independent of the underlying technology and it is conceivable that other solutions besides CDE's may emerge over time. Nevertheless, CDE's in their current form offer a functioning solution to meet these particular needs of developers.

5.2 Underlying Developer Pain Points

One of the primary motivations for a simple setup process and cross-environment compatibility that was mentioned in the literature, was developer happiness and productivity. Many sources contained powerful language that was used to express pain points that developers had experienced in setting up environments, installing dependencies and dealing with the "works on my machine" syndrome.

This implies that installing dependencies and maintaining consistent environments are real problems that are experienced passionately by developers. This observation is independent from CDE's and points to a broader issue with real impact on developer experience.

The results of the case study support this observation, as the difficulty of maintaining and setting up a complex environment was the primary initial motivation for creating the CLI tool which lead to the creation of the containerized development environment. Interviewees had also brought up past experiences of dealing with compatibility issues in previous projects, further indicating the existence of these underlying pain points.

The interviewed developers' language was not as strongly worded as in the literature, but most interviewees used different emotive expressions to indicate experienced pain and difficulty with environment setup. The case study also brought up that starting and managing multiple services is experienced to be difficult. In addition to simplifying the setup, the CDE would also appear to help developers manage the environment during development.

The identified developer pain points seem real and warrant further investigation. An improved understanding of the impact and pervasiveness of these problems could help us evaluate the value that solutions like containerized development environments provide to software development teams and other stakeholders.

5.3 Developer Onboarding

Many sources in the literature review expressed the idea of a simple setup process being beneficial for the onboarding of new developers to projects. However, only one source expressed an experienced benefit in facilitating onboarding with the help of containerized development environments. The manager who was interviewed for case study expressed a similar interest, particularly for facilitating a workflow and onboarding process that would enable new developers joining the project to be

productive from day one.

The case study covered the experience of one developer being onboarded to the case project with the containerized development environment. When compared to previous projects, the CDE was reported to have successfully offered a faster and more painless experience of setting up the environment. The interviewee pointed out, that having a Docker course available when he joined the project helped with the use of the containerized development environment.

The lack of evidence supporting this hypothesized benefit coupled with the value of its potential realization, indicates an open and interesting area for future research. Particularly, based on the case study it would appear that providing the necessary learning materials and training is an important factor in onboarding to projects using a containerized development environment. However, the same could be true of software projects in general, and is not inherently tied to containerized development environments.

5.4 Disposable Environments and Project Switching

Some sources in the literature review viewed the simple setup concept from a different angle, viewing CDE's as easily disposable environments. Developers that worked on multiple projects appreciated the ability to easily switch between project environments that were independent from one another.

In addition the ability to easily dispose of and setup a new clean development environment was valuable in situations where the environment ends up in a corrupted state. Both of the mentioned factors appear to be beneficial for teams and developers who frequently switch between projects or experience the need to reset their development environments.

Similar thoughts were expressed in the case study from the viewpoint of isolating different environments. One of the interviewed developers brought up his experienced difficulty with managing multiple Python environments on his machine, and that isolating these development environments inside of containers was valuable. Another interviewee brought up the difficulty of having different versions of the same database running locally, and that isolating these external services into containers enables working on multiple projects with different version requirements of the same dependencies.

It would appear that project-level isolation is a feature provided by containerized development environments, but the benefit it provides is only experienced in specific contexts. For teams working on a single project on dedicated single-purpose development machines, project switching is something that might never occur, and the isolation provided by a containerized development environment would not provide any major benefit. To contrast this, developers who need to switch between developing different systems with complicated dependencies, containerized development environments could provide a crucial benefit.

5.5 Infrastructure Package Management and Microservices

An interesting and novel benefit that emerged in both the literature review and case study, was the use of CDE's and the Docker container platform for infrastructure package management. Most programming languages have dependency management tools that can be used for specifying and installing required libraries and dependencies to a project. The adoption of a containerized development environment enables something similar at the level of infrastructure and services, making it possible to provision databases, message brokers and other services with just a few lines of code and distribute them team-wide.

This significantly lowers the barrier of taking new technologies and services into use in projects, as was observed and reported by the developers interviewed in the case study. Sources in the literature review also pointed out that this enables the testing and profiling of different versions of services and configurations. This aspect of containerized development environments is something that especially service oriented projects and teams could benefit from greatly, and the case study provides some preliminary evidence of this.

Interviewees of the case team mentioned that retrieving the latest version of the containerized development environment from version control and running a single command was enough to setup and run new services and external dependencies that had been added to the system by other developers. The simplicity of specifying new services with a few lines of code was highlighted by many developers as a notable positive feature. All of these things provide evidence of this lowered barrier for adding new services.

The interviewees also explicitly linked the capabilities provided by this feature to the project's shift towards a microservices oriented architecture. The relationship between the two appears to be symbiotic — originally it was the increase in services that lead to the CDE, but later on the team's developers viewed the CDE as enabling and facilitating the continual move towards distributed services.

One interviewee pointed out that when new services were added to the system, their configuration in the containerized development environment required the team to consider how the different microservices and their dependencies would be composed and deployed to production. This further demonstrates the compatibility between the containerized development environment and systems using a microservices architecture. Interviewees often mentioned that at the point where a system has three to four services, they would consider the CDE as being necessary.

The value of something resembling an infrastructure package management tool for external dependencies appears to be highly ranked by developers. While the use of containers for defining the language runtimes of the SUD appears to divide opinions, we found no criticism towards running external dependencies like databases as containers. Choosing to containerize only external dependencies appears to be a safe alternative for getting some of the benefits of CDE's while avoiding many of their common pitfalls.

5.6 Shared Configuration and Tooling

Another experienced benefit brought up in both the literature review and the case study, was the ability to implement configuration and tooling that could be shared team-wide. Interviewees in the case study expressed that the explicit configuration of how ports and environment variables are configured in the system, also helps to understand the structure of the system under development and how the different components of the system are connected.

The case team had also added services like browser-based database management tools to the containerized development environment that were then distributed across the team. This illustrates that in addition to the dependencies of the SUD itself, containerized development environments can also contain additional tooling and configuration for development purposes.

One of the sources in the literature review mentioned a configuration that automated the execution of unit tests every time changes were made to the source code. The ability to distribute this kind of configuration makes it easier for teams to craft shared development environments that facilitate and enforce desired workflows.

5.7 Cross-Environment Performance Issues

One of the biggest challenges experienced with the adoption of CDE's that was expressed in the literature review was decreased performance of the system under development running on the developers' machines, which had a negative effect on developer happiness and productivity. There appears to be a trade-off between performance and consistency that has to be addressed and considered when adopting a containerized development environment.

Another challenge brought up in the literature review is the variation in performance of CDE's across different platforms and operating systems. While CDE's successfully enable cross-environment consistency, they do not ensure cross-environment performance due to poor performance on non-Linux platforms.

These issues with performance were not brought up in the case study. Possible explanations for this could be that the development machines used by the case team just happened to be well-suited for running the containerized development environment, or that the SUD itself happened to be lightweight.

Quite a few sources in the literature review also mentioned performance issues with volumes when sharing source code between the host and container, which is commonly used in the implementation of containerized development environments. While performance issues related to volumes were not mentioned in the case study, interviewees did bring up the general difficulty of dealing with Docker volumes in implementation.

The severity of the performance issues experienced by a development team using a containerized development environment will depend on various factors. It would be

wise for any team contemplating switching over to a CDE, to test its performance on their development machines before making the switch. The Docker platform is still evolving and it is possible that these performance issues will improve over time, but currently they are one of the major drawbacks of using a CDE and the trade-off has to be carefully evaluated.

5.8 Non-Trivial Construction

One of the biggest challenges that was mentioned by developers in both the literature review and the case study was the difficulty in constructing the containerized development environment. As mentioned by the interviewees of the case team, getting the development environment working and fixing all of the related issues took considerable time and effort. The fact that the team had attempted to construct a containerized development environment early on in the project and abandoned it supports the existence and size of this challenge.

Containers are a technology that can be used as a tool in constructing development environments but it is not their primary purpose and, for this reason, there is no single way of constructing them. As noted by all interviewees in the case study and various sources in the literature review, the construction requires skills and knowledge of the tools used. While the construction itself was experienced as challenging, many interviewees mentioned that the environment is not difficult to operate in daily usage.

The necessity for team-wide knowledge of Docker could prove to be beneficial in DevOps-oriented teams that already use containers for deployment — the introduction of containers into the development environment would force all developers to familiarize themselves with the technology. The case study offers some evidence of this, as most interviewees mentioned that the adoption of the containerized development environment had increased their knowledge of the technology. One interviewee also expressed this opinion explicitly, stating that it benefits the team at large when every developer has better knowledge of the tools used in production.

While the flexibility of the tool makes initial construction more challenging, it also allows for the freedom for developers to craft containerized development environments according to their specific needs and tastes. An example of this would be the previously mentioned environment that enforced a TDD workflow, or the development tools added by the case team. One of the sources also stated that through experience their team had given up on fully containerized development environments, instead opting to use containers only for databases and other required services.

The difficulty in construction does not appear to be purely technical. Some of the interviewees from the case team pointed out that the construction required grit and tenacity from the team, and an underlying commitment to fix all of the issues related to the containerized development environment. This seems to hint that there is also a motivational challenge related to constructing containerized development environments, and that their successful adoption requires a significant commitment

from the team.

These challenges in construction are something that teams contemplating switching to a CDE have to keep in mind. If the team does not see great value in its adoption, or if it is unable or unwilling to expend the time and resources to research and develop the CDE, then it may not be a viable option.

5.9 Resources and Support from Management

The difficulty in implementation introduces another challenge that is related to the resources it consumes. The case team that was interviewed used a significant amount of its time to implement the containerized development environment and to fix all of the issues related to it. Interviewees pointed out that in the context of the case project this was possible, since there was no significant pressure to deliver new features to customers. If the context of the team and project was more competitive with strict deadlines, this investment in resources could have proven to be a challenge. As one of the interviewees pointed out, in short-lived projects resources could be better spent elsewhere.

The interviewees also brought up that the implementation of the CDE required the kind of team and management that would push for and allow its implementation. The case team was self-organizing and the team's manager did not stand in the way of the team's move towards the containerized development environment, which would appear to be a critical precondition. If the team was operating within the context of an organization where this kind of decision required managerial approval, it is possible that the project would not have moved forward.

5.10 Local Development Tools and Changes in Workflow

Both the literature review and case study brought up experienced challenges related to using locally installed development tools with the containerized development environment. Interviewees suspected that most of the tools that exist are designed to be used with applications that are running directly on the local host machine. The fact that the system under development is running inside of containers adds a level of isolation between the host and the application's components, which requires additional configuration to get the tools working correctly.

This challenge is also closely related to other changes that happen to the development workflow when a containerized development environment is taken into use. Many development tasks that are done from the command line, like running tests, now have to be executed inside of containers and this change requires developers to change the way of working they are accustomed to. As evidence of this, the manager of the case team initially opted out from the CDE for this very reason, as he was used to his previous workflow and did not want to fix what wasn't broken. Another interviewee mentioned that his use of certain debugging tools had lessened after the

adoption of the containerized development environment, providing further evidence of this change in workflow.

While in theory using the containerized development environment could be optional for developers working on the same project, based on the case study we were under the impression that team-wide usage was implicitly enforced. The fact that the team built its end-to-end testing around the containerized development environment, further established its default position in the project.

5.11 End to End Testing

The case study brought up the benefit of using the containerized development environment as a way of setting up the system for end to end testing. This use of the containerized development environment did not come up in the literature review, but was frequently mentioned by different interviewees in the case study.

The case team leveraged the simple setup of the containerized development environment as a means of creating a production-like replica of the entire system that end to end tests could be run against. This is not exclusive to containerized development environments, and has more to do with container-based deployment and orchestration tools in general. The case team's use of the CDE for end to end testing still illustrates the potential of using containers and orchestration tools for testing purposes.

One interviewee also hypothesized that the repeatable automated setup of the system could be used for creating multiple instances of the system, that could then be tested in parallel. This feature in particular could enable running large suites of tests in a much shorter amount of time. This use of containers in testing appears to be a novel and interesting research topic of its own.

5.12 Use of Gray Literature

The choice of using gray literature as source material plays a significant role in this study. It has been suggested that systematic literature reviews in software engineering may fail to capture the current state of the field as they typically exclude forms of gray literature (Garousi et al. 2016). One set of guidelines for including gray literature and conducting multivocal literature reviews in software engineering is presented by Garousi et al. (2019). We will briefly compare the structure of our study to this set of guidelines.

The review of gray literature in this paper was conducted using the standard systematic literature review protocol as a template which follows the first guideline presented by Garousi et al. The second guideline states that existing reviews on the topic should be identified first and the need for the review should be clearly motivated. Before conducting our review we searched for existing scientific literature on containerized development environments and were unable to find any papers

discussing the topic. The burgeoning interest towards CDE's expressed by practitioners in gray literature made us feel that there was a gap in the existing scientific literature for introducing the topic. The third guideline states that the decision to include gray literature should be made systematically, and while there was no systematic process behind us making the decision, our reasons for including gray literature in our review match many of the listed reasons suggested for this selection process by Garousi et al.

The fourth guideline states that the research questions should be defined so they address the goal of the review and match the needs of the target audience. One failure of this paper was to clearly communicate the existing gap in scientific literature and that the underlying goal of this paper was to introduce this recent phenomenon. However, the research questions were carefully formulated to achieve this goal by finding answers to why practitioners in the field were pursuing containerized development environments and what kinds of benefits and challenges they claimed to have experienced. The fifth guideline suggests to adopt various research question types which our study does not follow, as all of our research questions fall under the exploratory category.

The sixth and seventh guidelines state that one should identify the relevant types and sources of gray literature early on, and that general search engines and specialized websites are ways of searching for gray literature. Our study follows these guidelines as we identified a group of websites that contained the largest amount of quality discussion on the topic of containerized development environments. We used general web search engines to then target these websites specifically and crafted our search string so that it returned the key sources we had identified early on in the study. Our study also follows the eighth guideline that provides three different stopping criteria for gray literature searches. In our study we decided to define our stopping criteria by setting a boundary on the amount of effort that would be spent on reviewing the literature. In practice this meant that only the top 20 search results for each of the three chosen websites were checked against our inclusion and exclusion criteria. The ninth guideline states that quality assessment criteria should be mixed in with the inclusion and exclusion criteria. In our study the only requirement we specified for included literature was that the sources ought to contain recent and relevant information for the research questions.

Guideline 10 states there should be coordinated integration of source selection for both the gray and formal literature. No formal literature was included in the review of our study which means this guideline was not followed. Guideline 11 states that different types of criteria should be applied and adapted for the study quality assessment of gray literature. We did not perform any formal quality assessment for the sources of gray literature in our study.

Guideline 12 states that during data extraction explicit traceability information should be maintained between the extracted data and the primary source. In our data extraction form the explicit link between quotes and their sources was captured and this link was maintained all the way through the data synthesis phase as well.

Guideline 13 states that a data synthesis method suitable for the source of literature should be selected. In our case, we considered qualitative coding to be particularly suitable for synthesizing the self-reported claims of developers captured in the data extraction phase.

Guideline 14 states that the writing style should match the target audience. Our study adheres to this guideline as the methodology behind our study is transparent and has been clearly documented, the key findings of our study have been highlighted, and potential directions for future work have also been identified.

Our review and use of gray literature follows a vast majority of the guidelines presented by Garousi et al. (2019). The two major deviations from these guidelines were (1) solely relying on gray literature due to the lack of existing scientific literature and (2) not using exhaustive quality assessment of the included sources of gray literature.

5.13 Threats to Validity and Limitations

The literature review relies exclusively on self-reported postings acquired from online forums and interviews, although forum discussions generally contained more debate providing some variety within the single category. The fact that the data was sourced exclusively from online forums may have biased the sample, since it is possible that only a particular segment of developers discuss topics actively online. As a result, any dominant traits or opinions held by this segment would be heavily featured and over-represented in the results of the literature review. In this way a highly enthusiastic and optimistic segment of developers could create an overly optimistic view of the trend and vice versa. It is also possible that this segment represents some bubble that holds certain beliefs and opinions as undisputed fact. If this were the case it could inhibit certain issues from appearing in the data.

Another threat to validity is that the concept of a containerized development environment is loose in definition and refers to a variety of different kinds of implementations. This means that differences in experience may have resulted due to a difference in implementation. In this early stage of an emerging trend where there is no canonical model for the concept this is an unfortunate but unavoidable reality.

All extracted citations in the literature review came from developers working on different projects and teams. Their self-reported experiences may represent a fringe opinion that paints an overly positive or negative picture of the actual experience. For this reason we tried to gather a large sampling of opinions from different projects, hoping to average out and mitigate the impact of fringe opinions.

While the case study offers multiple accounts from different developers working on the same project, the gathered interview data is still solely self-reported. Thus, any conclusions about actual measurable effects that the containerized development environment had on things like productivity, cannot be made.

We also hold an acknowledged personal bias in favor of containerized development

environments, which may have skewed various areas of the study. However, active measures were taken to counteract this acknowledged personal bias. Examples of this include having the interview questions reviewed by a senior researcher, and including an explicit research question that covers the negative experiences of developers.

The fact that the interviews were conducted by a member of the case team may have had an impact on the quality of the collected interview data. It is possible that due to social, political, or other reasons of this nature, the interviewees may have been selective about the stories and opinions they shared during the interview. As an example, developers may have chosen not to share their negative experiences in order to avoid conflict or they may have attempted to answer the questions in a way that would please the interviewer.

It is likely that the structure of the study influenced the results of the case study. The literature review was conducted before the case study by design. Knowledge of the concepts from the literature review may have lead to an unconscious attempt to find the same patterns and concepts when analyzing the interview transcripts, and have caused us to overlook concepts that were missing in the literature review.

With these limitations and threats to validity in mind, it is worth noting that active measures were taken in various parts of the study's design in an attempt to pursue as neutral a review as possible of containerized development environments from the perspective of individual developers. Much time and effort went into considering what our choice of gray literature and interview transcripts could be considered compelling evidence of, which is reflected in the scope of our research questions and the title of this thesis. The large collection of self-reported claims that were analyzed in this study reflect the real experiences and feelings of developers who have used containerized development environments in their own work. The fact that we were able to identify a variety of concepts and captured an even mix of both positive and negative experiences leads us to believe that we have managed to make meaningful observations and findings on this topic from the chosen perspective.

6 Conclusions

In this thesis, we investigated the motivations and experiences of developers who have adopted containerized development environments in their projects. The study was carried out as a combination of a systematically conducted review of gray literature and a qualitative single-case study. Due to the recent nature of the trend as well as the suitability to answer the research questions, the material for the literature review was purposefully sourced from blog posts and forum messages submitted to websites frequently used for discussing the topic. The case team and project were purposefully selected due to the practical relevance of the case, as well as convenient access to the team's members. The data for the case study was collected through semi-structured theme interviews that were transcribed, analyzed, and then compared against the results of the literature review.

The results of this thesis indicate a common motivation to mitigate cross-platform consistency issues and to provide a simple automated setup for development environments in order to boost developer happiness and productivity, paving the way for efficient developer onboarding. The results of both the literature review and case study confirm the experienced benefits of cross-platform consistency and simple environment setup. Little evidence of experienced benefits with developer onboarding was found in the reviewed literature, but the experience of a single developer in the case study appears to indicate a positive and efficient onboarding experience, with the added requirement of providing the necessary learning materials for Docker and containers. The overall lack of evidence points towards a potential avenue for future research.

The results of both the literature review and case study also indicate a novel use of the Docker container platform as an infrastructure package manager, making it possible to provision databases, message brokers, and other services to projects with just a few lines of code and distribute them team-wide. This appears to successfully lower the barrier of introducing new services and dependencies to the project and has implications that appear to be exciting for developers working on service oriented projects. The case study indicates a particular suitability of the containerized development environment for teams migrating towards a microservices architecture largely due to the combination of shared configuration management and this feature.

The results of the literature review indicate an apparent trade-off between cross-platform consistency and performance — the decreased performance with containerized development environments is one of two major challenges experienced by developers. Performance on non-Linux operating systems in particular suffers due to the technical nature of containers. However, this decrease in performance was not evident in the case study, as the developers of the case team worked exclusively on Linux workstations.

The results of the literature review and case study also indicate that the implementation of containerized development environments itself is a challenge and requires knowledge of containers and related technologies in order to be executed successfully.

The technologies themselves, however, are flexible and allow skilled developers to customize containerized development environments to match their particular needs. Introducing containers to the development environment can also be beneficial for companies seeking to foster a DevOps culture by acquainting developers with containers in their daily practice — this is supported by the findings of the case study.

The case study also reveals a motivational challenge related to implementation. The successful adoption of a containerized development environment requires a significant commitment from the team to overcome the difficulties faced during implementation. The interviewees also pointed out that due to its difficult and time-consuming nature, the implementation of a containerized development can be a drain on the development resources of the team. The case study also implies that certain preconditions are required from the team's management and the team's structure in order to facilitate the implementation of the containerized development environment.

The case team that was studied also exhibited a novel use of leveraging containers and related technologies in the implementation of end to end tests for complex systems. The case team used the containerized development environment as a way of setting up a production-like system that tests were run against. Members of the team also hypothesized that orchestration tools could potentially be used for running multiple large test suites in parallel, significantly cutting down on test execution time. This use of containers in testing appears promising and points to another area for future research.

Containerized development environments offer many highly desirable benefits but they come at a cost. The findings of this thesis indicate that future research into CDE's is warranted and there are many interesting avenues for research available. The motivations, benefits, and challenges found in this thesis could serve as a starting point for future investigation in other case studies. In addition, researching the effects that CDE's have on the onboarding of new developers appears to be of great interest for both developers and managers, and is another potential topic for future research.

On a more general level, this study indicates that the underlying issues that containerized development environments attempt to solve are of real impact and importance to developers. There is an evident desire and need for technologies that will help developers battle the "works on my machine" syndrome, to provision infrastructure and services effortlessly, to automate environment setup, and to deal with distributed configuration management at the level of the entire system under development. We believe that each of these issues warrants further investigation and that there is room for new tools and solutions to help solve these issues.

References

Anderson, C. (2015). Docker. *IEEE Software*, *32*(3), 102–105.

Continuous Integration. (2006). Retrieved April 25, 2019, from https://martinfowler.com/articles/continuousIntegration.html

Davis, J. & Daniels, R. (2016). *Effective DevOps: Building a culture of collaboration, affinity, and tooling at scale*. O'Reilly Media, Inc.

Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous integration: Improving software quality and reducing risk*. Pearson Education.

Erfani Joorabchi, M., Mirzaaghaei, M., & Mesbah, A. (2014). Works for me! characterizing non-reproducible bug reports. In *Proceedings of the 11th working conference on mining software repositories* (pp. 62–71). MSR 2014. Hyderabad, India: ACM. doi:10.1145/2597073.2597098

Fink, J. (2014). Docker: A software as a service, operating system-level virtualization framework. *Code4Lib Journal*, *25*, 29.

Garousi, V., Felderer, M., & Mäntylä, M. V. (2016). The need for multivocal literature reviews in software engineering: Complementing systematic literature reviews with grey literature. In *Proceedings of the 20th international conference on evaluation and assessment in software engineering* (p. 26). ACM.

Garousi, V., Felderer, M., & Mäntylä, M. V. (2019). Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, *106*, 101–121.

Humble, J. & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation (adobe reader)*. Pearson Education.

Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is DevOps?: A systematic mapping study on definitions and practices. In *Proceedings of the scientific workshop proceedings of XP2016* (12:1–12:11). event-place: Edinburgh, Scotland, UK. New York, NY, USA: ACM. doi:10.1145/2962695.2962707

James Shore: Continuous Integration is an Attitude, Not a Tool. (2005). Retrieved April 25, 2019, from http://jamesshore.com/Blog/Continuous-Integration-is-an-Attitude.html

Jaramillo, D., Nguyen, D. V., & Smart, R. (2016). Leveraging microservices architecture by using docker technology. In *SoutheastCon 2016* (pp. 1–5). SoutheastCon 2016. doi:10.1109/SECON.2016.7506647

Khan, A. (2017). Key characteristics of a container orchestration platform to enable a modern application. *IEEE Cloud Computing*, *4*(5), 42–48. doi:10.1109/MCC.2017.4250933

Manage data in Docker. (2019). Retrieved April 25, 2019, from https://docs.docker.com/storage/

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, *2014*(239).

Meyer, M. (2014). Continuous integration and its tools. *IEEE software*, *31*(3), 14–16.

Overview of Docker Compose. (2019). Retrieved April 25, 2019, from https://docs.docker.com/compose/overview/

Rajkumar, M., Pole, A. K., Adige, V. S., & Mahanta, P. (2016). DevOps culture and its impact on cloud delivery and software development. In *2016 international conference on advances in computing, communication, automation (spring)* (pp. 1–6). 2016 international conference on advances in computing, communication, automation (spring). doi:10.1109/ICACCA.2016.7578902

Rogers, R. O. (2004). Scaling continuous integration. In *International conference on extreme programming and agile processes in software engineering* (pp. 68–76). Springer.

Spinellis, D. (2012). Virtualize me. *IEEE software*, *29*(5), 91–93.

Webster, J. & Watson, R. T. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly*, xiii–xxiii.

What is a Container? (2019). Retrieved April 25, 2019, from https://docker.com/resources/what-container